超算环境科学工作流应用平台的引擎设计和资源调度 *

李于锋 [1]，莫则尧 [1,2]，肖永浩 [1†]，赵士操 [1]，段博文 [1]

(1. 中国工程物理研究院 算机应用研究所，四川绵阳 621900; 2. 北京应用物理与计算数学研究所，北京 100094)

摘　要：高性能计算机体系结构的复杂性对使用者提出了更高要求，而且在工程实际和科学实验中，通常需要使用多种应用软件相互协作才能解决复杂问题。围绕超算资源的易用性和多类软件的集成以及协作需求，开发了超算环境下的科学工作流应用平台，设计了异步并发的流程执行引擎，采取调度算法和调度器、引擎相分离的设计策略，给出了资源调度方案。提出了局部资源池化技术和资源预约算法，并比较分析了五种常用调度算法的性能，给出了算法选择的建议。实际应用表明设计的引擎能够支撑复杂工作流的灵活执行方式，给出的资源调度方案能够满足超算环境下工作流应用的高效执行。

关键词：科学工作流；高性能计算；资源调度；工作流引擎

中图分类号：TP301　　doi: 10.3969/j.issn.1001-3695.2018.05.0263

Engine design and resource scheduling of scientific workflow application platform in supercomputing

Li Yufeng[1], Mo Zeyao[1, 2], Xiao Yonghao[1], Zhao Shicao[1], Duan Bowen[1]

(1. *Institute of Computer Application*, *Chinese Academy of Engineering Physics*, *Mianyang Sichuan* 621900, *China*; 2. *Institute of Applied Physics &Computational Mathematics*, *Beijing* 100094, *China*)

Abstract: The rapid development of HPC hardware gave rise to high complexity of high performance computer hierarchy, which made it hard for users to make full use of HPC resources. In general, lots of software need to be cooperated together for particular object in scientific experiments and engineering domains. This paper described a new scientific workflow application platform in HPC environment. This platform contained an engine with high concurrency and asynchronous framework to process workflow application. Scheduler, planner and engine were decoupled from each other, which allowed that the three components can develop independently: scheduler for scheduling algorithms implementation, planner for collecting the scheduling information and engine for workflow driver. Scheduler and planner used resource advance reservation and local pooling mechanism to increase the performance of workflow execution. This paper also implemented and compared five scheduling algorithms as to their performance on testing graph set, and got some useful advices of algorithm selection. Real application show that engine can support various execution strategies and resource scheduling solution can help increase efficiency of workflow execution in supercomputing environment.

Key words: scientific workflow; high performance computing; resource scheduling; workflow engine

0　引言

数值模拟在科学实验或者工程实际中发挥着重要作用。随着高性能计算机的快速发展，复杂异构体系结构成为主流，如TOP500[1]排名前列的并行机均使用了各种加速设备和多套集群系统。在开发软件时，需要开发者考虑多级嵌套并行和能耗问题。在使用 HPC 应用软件时，仍然需要用户熟悉计算环境，才能更好地结合软硬件的优势。在科学发现和工程实际中，为了解决一个实际问题，通常需要使用一系列相关软件，典型的流程是建模、模拟、可视分析，它们之间可能有数据的依赖关系，这些软件相互协作才能给出最终结果。2015 年美国发布了创建国家级战略计算发展规划（NSCI）总统令[2]，其中明确要完成的第二个目标就是提升用于实现建模与模拟以及数据分析计算所需之各类技术基础之间的连贯性。科学工作流是解决此类问题的关键技术之一，在超算环境中，利用科学工作流技术将超算资源环境透明化，为用户屏蔽作业投递和数据管理等细

收稿日期：2018-05-09；**修回日期：**2018-06-21　　**基金项目：**国家重点研发计划资助项目（2016YFB0201504，2018YFB0703903）

作者简介：李于锋（1982-），男，河南光山人，高级工程师，硕士，主要研究方向为科学工作流、高性能计算（liyf@caep.cn）；莫则尧（1971-），男，研究员，博士，主要研究方向为高性能计算、应用软件框架研制；肖永浩（1981-），男（通信作者），高级工程师，博士，主要研究方向为连贯性计算；赵士操（1986-），男，工程师，硕士，主要研究方向为集成业务平台软件架构设计；段博文（1990-），男，工程师，硕士，主要研究方向为 Web 应用开发．

节，并提供简捷的流程组装和运行界面，能极大地促进超算环境和应用软件的协同发展。

科学工作流是在复杂科研过程中，对数据收集、处理、分析和可视化等步骤进行按需组合，并进行服务选择和资源映射匹配，自动在分布式异构资源环境下按照自定义逻辑关系顺序进行执行的流程。肖飞等人[3]将工作流的发展分为三个阶段：以集成化软件包为主要体现的萌芽期、商业工作流、科学工作流。科学工作流区别于传统的业务工作流的特点主要是其流程复杂，面向数据，且要具有动态适应性。国内外已经出现了很多工作流平台，比如美国加州大学研究者开发的 Kepler[4]系统，该系统起源于 2002 年，基于加州大学伯克利分校的 Ptolemy II 系统开发。Kepler 是一个活跃的开源项目，目前已发布 2.5 版。包括生态学、分子生物学、化学、计算机科学、电子工程和海洋学等领域在内的很多研究者对 Kepler 做出了贡献。美国南加州大学（USC）信息科学研究所（ISI）开发的系统 Pegasus[5]，可以将抽象工作流映射到具体执行环境中并执行，提供的技术可以将工作流部署到不同的环境下：包括桌面环境、校园集群，网格和云。截止 2018 年 5 月，发布最新版本 4.8.2，该系统已应用在 LIGO 项目中辅助引力波探测分析。Taverna[6]是英国曼切斯特大学计算机科学学院开发的工作流套件，起源于 myGrid 联盟，myGrid 旨在实现 e-Science 和 e-Laboratoies，由多个组织合作，并发起了很多项目，为科学家进行电子化科研提供很多实用工具，很多已经广泛应用于生物科学、社会科学、化学，天文学、音乐以及多媒体等领域。Taverna 是其中一个工具，用来设计、编辑、执行科学工作流。它将分布式网络服务和本地工具进行管道式组装以便完成复杂分析过程。不仅可以本地桌面环境执行，还可以在大型基础设施如超算环境，网格和云计算环境上执行。全世界包括学术的和商业的 350 多个组织在使用 Taverna，为他们的创新研究加速。王红霞[7]基于 GT4 网格环境设计了工作流引擎，但关注点在网格服务及其协作关系，并非本文关注的科学工作流系统。沈瑜等人[8]设计和实现了一种高性能资源统一管理系统，关注于多套高性能资源的统一管理和审计，其中针对用户的资源预分配和定时审计机制能更充分利用资源，本文提出针对工作流的资源预约和池化策略能更好的满足工作流应用的高效执行需求。

已有的科学工作流系统普遍存在的问题有：任务粒度太细，甚至简单的排序和代数运算都被设置成组件，造成使用繁琐、复杂；对底层系统的配置复杂，造成部署代价高；大部分工作流运行依赖运行系统的调度方法，没有针对流程的调度策略，造成调度效率低下等问题。

中国工程物理研究院计算机应用研究所的 HSWAP 平台以高性能应用软件作为组件封装的基本单位，配置简单，部署容易，支持跨平台应用的协作。调度采取了资源预约和局部资源池技术，支持多种资源调度算法，具有流程级别的 DAG 调度算法，可有效提升流程执行效率，是集成多类软件在 HPC 环境下数值模拟和数据分析的有力工具。

在科学工作流的应用过程中，一般分为静态的编排阶段和动态的运行阶段。编排阶段解决工作流的用户自定义问题，包括流程编辑，组件属性编辑等；运行阶段包括执行工作流引擎，驱动流程运行，以及组件任务在 HPC 资源上的调度和任务状态监控等。本文重点阐述运行阶段工作流引擎的设计和资源调度方案。

1　HPC 科学工作流应用平台 HSWAP

1.1　HSWAP 简介

HSWAP 是中国工程物理研究院计算机应用研究所超算应用团队开发的超算连贯性计算平台，旨在 HPC 环境中使用科学工作流技术提供集成的超算服务模式助力科研人员提高工作效率。在科学工作流中，一个作业是指一个完整流程；一个作业由若干任务组成，每个任务在 HSWAP 中被封装为一个执行组件，代表一个建模、模拟或可视分析的任务。组件可以由用户自定义属性，比如输入输出设定等。组件之间可以有数据的依赖关系。流程可以用有向无环图 DAG 来表示，记为 G=(V,E)：其中 V 表示任务组件的集合，E 表示任务结点的依赖关系集合，有向边表示任务间的依赖关系，边起点任务称为终点任务的前序任务结点，边终点任务称为起点任务的后继任务结点。

HSWAP 的界面如图 1 所示，界面左侧展示组件库，通过拖放的方式在流程图中添加实例；界面右侧为流程编排与展示区，通过拖拽方式添加组件实例，双击鼠标编辑组件属性，以鼠标进行组件间划线定义组件的依赖关系。在顶部配有控制运行的按钮，在运行阶段，可以不同颜色展示运行状态。切换到应用标签页时可实时与组件应用图形界面进行交互。



图 1　HSWAP 的界面

1.2　HSWAP 的架构设计

为支撑不同领域的数值模拟服务和产品设计流程，集成各类应用场景的应用技术共性，设计的 HSWAP 整体架构如图 2 所示。采用 B/S 模式实现，用户端只需要有网页浏览器即可使用该平台，具体的业务逻辑在服务器端实现。部署十分便捷，只需部署一份服务器端平台软件而无须部署客户端，即可以多用户同时使用。

控制层负责工作流设计 API 接口和工作流执行 API 接口，主要用来连接用户在图形化的浏览器客户端进行可视编辑和运行控制与后台的命令实现。灵活而强大的 API 接口设计保证了平台的稳定性。业务逻辑层则描述了各行业领域内的业务逻辑，比如结构力学领域的建模-模拟-可视分析流程，在其他领域比

如生物、材料领域，同样有类似的流程，只是流程的结构可能不尽相同。基础服务层和数据层是平台的核心实现，基础服务层提供了组件管理、流程创建、流程执行引擎和资源调度、应用代理等功能，数据层则管理各类元数据。HSWAP 的特色功能包括远程应用软件界面推送至内嵌浏览器客户端支持实时交互，同一工作流实例支持跨平台（Windows、Linux 等远程系统）的软件应用组件化配置和协作运行，灵活的运行控制策略等。

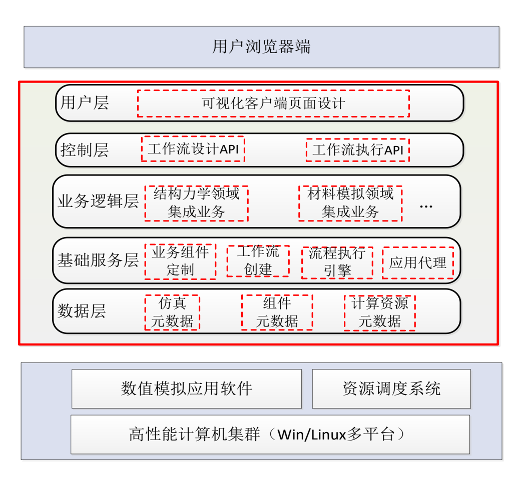


图 2 HSWAP 平台的架构设计

2 异步并发的工作流引擎设计

平台的科学工作流以有向无环图表示。工作流引擎设计的目的是提供流程的执行逻辑，驱动流程运行，监控流程中任务组件的运行状态，及时反馈用户的操作指令。引擎与其他模块之间的关系如图 3 所示，用户层给引擎发送启动项目、改变运行策略、暂停、终止工作流运行等指令，引擎响应相关命令，进行相关操作，并将流程中任务状态及时反馈给用户。如用户启动项目，则引擎会从流程创建数据库取得抽象工作流，进行流程解析和执行。执行任务组件时，将子任务投递到高性能计算机上的应用代理服务器，应用代理会监控子任务运行状态，将状态及时反馈给引擎。工作流中任务的依赖关系由引擎负责解释和处理，相关数据的复制和迁移也首先由引擎触发，然后交给数据管理模块处理。

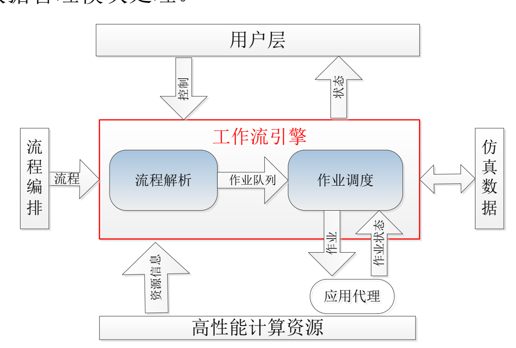


图 3 工作流引擎与其它模块的联系

由于引擎一方面要负责监控任务组件的执行，另一方面要及时处理用户在前端的操作指令，所以引擎的设计必然要采用并发的模式，又由于用户操作和组件状态改变等存在紧密相关性，而某些操作所需时间较长，为提高响应时间，改善用户体验，故采用异步模式设计。异步并发的工作流引擎由一个事件循环主体构成，图 4 是其基本运行架构的示意图，围绕一个工作流实例，进行 DAG 拓扑排序、可投作业产生、资源调度、作业投递等一系列步骤。在各个阶段之间都可响应外部 API 调用，包括运行控制 API 和作业状态设置 API，前者由用户界面前端触发，后者由服务端应用代理触发。工作流引擎及时响应外部 API 调用，根据当前作业运行状态和流程状态决定采取相应的动作。

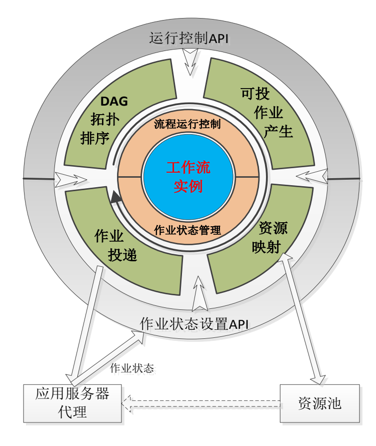


图 4 工作流引擎的运行架构

引擎由 Python 编码实现，借助异步 IO 实现事件循环功能和高性能 RPC 库实现外部 API 调用接口，将引擎与平台的其他模块进行解耦，具有独立运行的特性，可作为独立进程进行部署，并支持并发部署，以支持多工作流多引擎并行执行。

3 基于工作流的资源调度

3.1 HSWAP 资源调度方案

在超算环境中，作为系统软件都部署有一套作业调度系统，比如 PBS、LSF 或者 SLURM 等，用来进行作业调度，为计算任务分配所需资源。一般情况下，用户提交作业相互独立，被调度系统分别调度，分别经历投递、排队、启动运行、运行、结束等状态。在科学工作流的运行环境中，处于一个工作流作业中的任务具有相关性，它们之间可能是并行的，也可能是串行的数据依赖或控制依赖关系。若不考虑这些关系，单独将每个任务直接投递到超算环境的调度系统中，则每个任务都可能经历较长的排队等待时间，而且后续任务只有在其前序任务完成后才开始投递，造成流程整体执行时间过长，效率低下的问题。

在科学工作流的应用中，由于调度时已知任务的依赖关系，如何充分利用这些关系提高调度性能值得研究。在 HSWAP 中，通过资源预约和池化技术以及高效的 DAG 调度算法保证工作流的高效运行。

HSWAP 的资源调度过程如图 5 所示。其中应用代理的目的是提供任务执行和监控的统一接口。平台基于应用代理和高性能计算软硬件环境建立了一个 HSWAP 的动态资源池，为调度器 Scheduler 配置支撑调度算法库（图中①步）。配置后用户的工作流任务调度时使用动态资源池中的资源，调度器则使用

调度算法库中的调度算法。当用户执行工作流引擎时，引擎将用户工作流完整信息传递给调度规划器 Planner（图中②步），调度规划器从动态资源池获得可用资源，进行资源筛查，将符合条件的资源和工作流信息传递给调度器（图中③-⑤步）。调度器调用具体算法，将调度结果传回给 Planner（图中⑥步），规划器将结果传回给工作流引擎（图中⑦步）。工作流引擎根据调度结果将任务通过应用代理投递到相应的超算资源（图中⑧-⑨步）。最后，应用代理在实际环境中启动任务运行并监控运行状态。

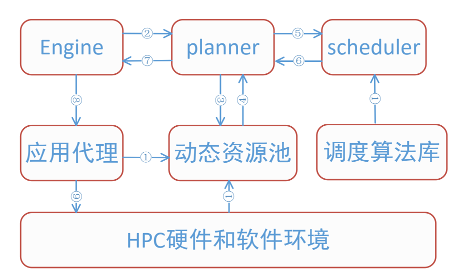


图 5　资源调度过程

3.2　资源池化与预约算法

实现资源池化管理，对工作流运行所需的资源进行提前预约是缩短工作流作业中所有任务整体完成时间、提高工作流执行效率的有效途径。

为支持工作流中多分支并发执行，需要预约充分的资源以提高并发性；但从资源利用率的角度考虑，为避免资源闲置和资源浪费，最好的方式是工作流中任务之间能够复用资源，特别是有前后顺序约束的任务之间，以期使用最少数目的资源即可满足需求。

若以整体预约资源建立局部资源池，提前获得资源控制权则能够满足工作流无等待延迟的执行。在实际环境中通常不能满足需求，这时在可用资源中预约尽量接近整体预约池大小的资源，在执行过程中动态改变资源池，达到动态调度的目的，相关过程如图 7 所示。

```
整体预约算法（make_plan）:
输入：工作流程图 DAG，图中任务组件 t 的资源需求描述 req(t)，可用资源集合 R
输出：工作流整体需要预约的资源集合 Req_all
1）对每个组件 t，计算满足该组件调度需求的资源集合 eligible(t)；
2）计算流程的 DAG 拓扑排序；
3）设置辅助数组 avail(t)，记录每个组件执行完毕后释放的可用资源集合；
4）for t in 拓扑排序后的组件列表：
5）  for pre_t in 组件 t 的所有前驱组件列表：
6）    res_from_ancestor = avail(pre_t) ∩ eligible(t)；
7）    if res_from_ancestor 不空：
8）      按 req(t) 中各类别资源需求数量，从 res_from_ancestor 中尽可能多的分配，
   之后，更新 req(t) 中资源数量；  更新 avail(pre_t)中剩余可用资源数量；
9）       if req(t) 为空，则分配成功，记录下 avail(t)为已分配资源，转 4）；
10）   Req_all = Req_all + req(t)中资源需求量，更新 avail(t)数组。
```

图 6　HSWAP 的资源预约算法

3.3　调度算法分析

由于调度问题具有 NP-难的复杂度，大部分研究集中在启发式算法的提出和改进上，还有很多研究集中在网格和云等分布式计算基础设施上考虑成本优化以及具备容错功能的动态自适应调度。Min-Min 算法是最简单常用的启发式算法，思想是最快完成的任务优先调度；Max-Min 算法是在每个任务的最小完成时间（对所有可用资源）中选择最大的那个任务来调度。Min-Min 算法和 Max-Min 算法没有考虑 DAG 图本身的特征，只是对就绪任务和可用资源进行局部决策。Topcuoglu 等人[9]于 2002 年提出 HEFT 算法，使用 DAG 图中任务节点的 upward rank 值作为优先级进行就绪任务的调度，取得很好的效果，是常用的启发调度算法，自提出后一直被作为算法比较的标杆。Shi 等人[10]建立了工作流的有向无环图表示和资源的无向有权图表示，并建立了任务间数据通信的模型，在处理器能力有显著差异的情况下改进了 HEFT 算法，提出 AEFT 算法。Kwok[11]从关键路径的动态性出发提出 DCP 算法，采用动态关键路径和插入时间槽的方式进行调度，Rahman[12,13]将动态关键路径算法 DCP 扩展到异构计算环境，并且在文中给出了多种经典算法的详细描述。1997 年 Chan 等人[14]针对异构环境提出了一种复杂度低的 HDSC 算法,该算法源自 1994 年 Tao Yang 提出的 Dominant Sequence Clustering 算法，思想是针对调度图的关键路径进行聚类，进而优化调度方案。George[15]针对网格上工作流的调度，以最少完成时间为目标提出了 MMGWS 算法，研究使用了提前预留资源的机制。Patil 等人[16]使用了 Maui 调度工具和 SLURM 资源管理和调度系统研究了在 HPC 环境下的考虑能耗的调度方法。Chen 等人[17]研究了在预留机制下，后续作业的调度如何影响先前的预约资源而进行自动调整，并考虑到任务实际执行时间与估计不符造成的相关问题。Wu 等人[18]对云计算环境下的工作流调度进行了综述，介绍了静态调度和动态调度，以及静态计划和动态调度相结合的方法等。

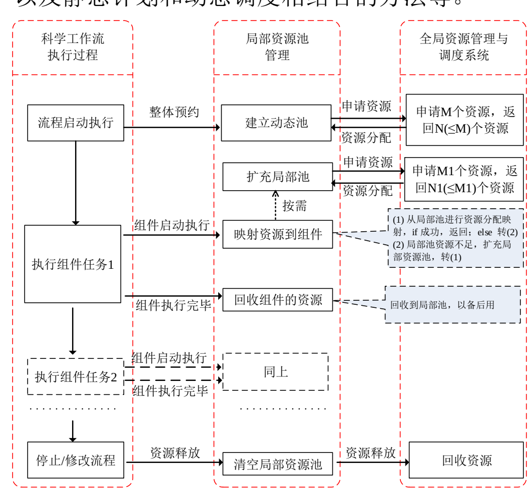


图 7　资源池化管理与工作流执行

如今的超算中心或数据中心都具有多套计算资源系统，异构特征明显。工作流中任务对资源的需求也各不相同。本文对常用的 Min-Min, Max-Min, HEFT, HDSC, DCP 等五种启发式算法进行了异构环境下的实现与性能测试，分析了它们在分布式异构特征资源上进行多种类型 DAG 调度的性能。

针对不同的 DAG 特征，本次实验选取了任务数和计算通信比两个特征，每种组合随机生成 25 个 DAG，每种算法对这些图进行平均性能比较。判断标准为 NSL（normalized scheduling length）,即以 DAG 的关键路径在最快资源上的执行时间为基准，NSL 定义为算法对图的调度长度与图的基准值之比值。显然，NSL 的最小值为 1，值越小则算法性能越好。模

拟资源环境为两个 cluster 组成的混合资源环境，每个 cluster 有 5 台机器组成，性能各异，cluster 内和 cluster 之间的机器通信带宽不同。

图的大小（任务个数）为 10~100，间隔 10 取值，共 10 种情况。通信计算比 CCR 分为三种类型，设任务处理的数据规模为 N，输出数据字节量为 $8N^2$，则类型 I 定义的任务计算复杂度为 $O(N^2)$；类型 II 定义的任务计算复杂度为 $O(N^2\log_2(N^2))$；类型 III 定义的任务计算复杂度为 $O(N^3)$。类型 I 的数据通信量相对计算量较大，类型 III 则较小。对三种 CCR，实验模拟结果如图 8~10 所示，其中模拟 DAG 数据来自随机图生成工具 daggen[19]，资源环境借助 SimGrid[20]工具模拟，SimGrid 是一个通用的分布式系统模拟器，可模拟网格和云资源环境，也可以模拟 MPI 程序环境和高性能计算资源系统。采用 C/C++语言实现了五种调度算法 Min-Min, Max-Min, HEFT, HDSC, DCP。基于 SimGrid 提供的资源环境表示以及主机和任务的封装 API，测试了五种算法在各类 DAG 工作流程图上的调度性能。

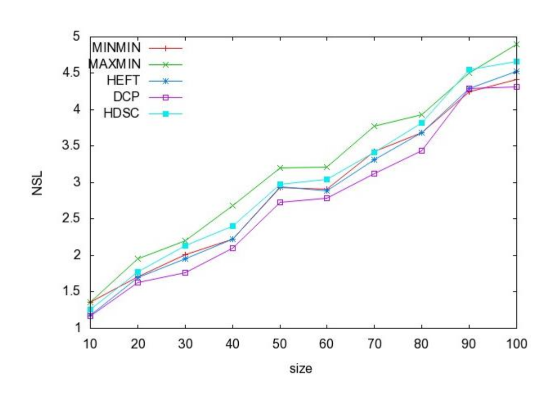


图 8　CCR 为类型 I 时算法性能

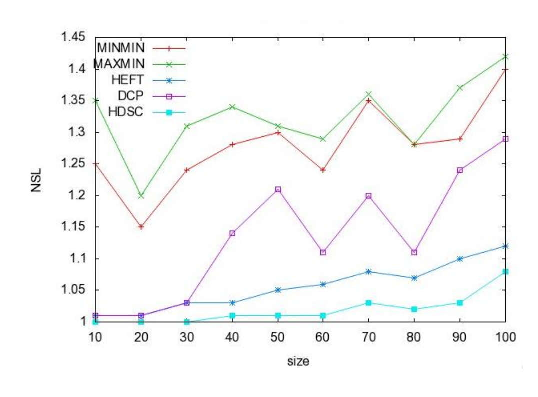


图 9　CCR 为类型 II 时算法性能

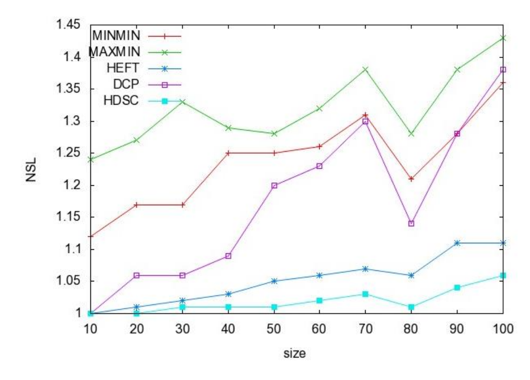


图 10　CCR 为类型 III 时算法性能

从图中可以得出如下结论：a）HEFT、DCP 与 HDSC 算法表现优于 Min-Min 和 Max-Min 算法，表明考虑了关键路径的算法性能更好，HEFT 算法平均性能较好，证实了其被广泛使用的原因；b）在计算密集型应用中（如图 9-10），HDSC 算法表现最好；c）DCP 算法适用于通信量较大而计算量小（如 CCR 类型 I）的 DAG 调度，而对于通信量小而计算量比较大的情况，对于大规模工作流调度性能略有下降。另外图中曲线整体均呈现上升趋势（虽有波动），这是因为在大规模 DAG 工作流调度时，由于启发式调度局部决策，造成性能与理想的基准差距累计而来。图 8 中的 NSL 值一般大于图 9、10 中的值，这是因为图 9 的 CCR 类型 II 和图 10 的类型 III 对应于计算密集型工作流，通信数据量较小（或者等价的具有高带宽的通信资源）的情况；而图 8 的 CCR 类型 I 对应的数据传输较多或者通信量较大。而计算 NSL 所采取的基准并不考虑数据通信，只考虑关键路径上的计算任务在无限多具有最快处理器的资源上的执行时间，当数据传输较多或数据量较大时，实际调度的 NSL 必然增大。

HDSC 算法使用了回溯技巧和聚类技术，减少了数据通信时间，提升了调度效率。但其只采用了回溯一层和聚类相邻单亲父子任务，还有优化空间，这对于开发新的调度算法有启示意义。另外 HPC 科学工作流在大规模计算资源上的调度，可以借鉴在网格和云上调度的优秀方法和最新算法，比如资源预约计费策略和动态自适应调度等，这方面的研究将有效提升超算资源利用率和工作流执行效率。

4　结束语

本文介绍了超算连贯性计算平台 HSWAP 及其工作流引擎设计和资源调度方案。平台一方面为用户屏蔽高性能计算底层软硬件资源使用的复杂度；另一方面为用户提供自定义流程、流程模板、自定义任务属性等功能，极大地提高用户工作效率。异步并发的工作流引擎设计能同时满足用户交互控制和工作流运行时控制的需求。资源调度模块采用了资源池化技术，并采用资源调度算法与调度规划器、引擎分离的设计，有利于调度算法的新增扩展。在异构环境下实现了常用 5 种启发式调度算法，并测试分析了它们在 DAG 流程调度时的性能。目前实现的调度算法依赖于任务的估计时间，如何准确估计任务所需时间尚需进一步研究，例如开展基于性能历史数据库利用机器学习等技术给出任务运行时间估计的工作。调度方面值得研究的另一方向是任务预估时间误差对调度算法的影响和自适应调整策略。

参考文献：

[1] Top500. org. Top 500 list [EB/OL]. (2017-11-30) [2018-05-08]. https://www. top500. org/lists/2017/11/.

[2] Obama. NSCI executive order 13702 [EB/OL]. (2017-07-20) [2018-05-08]. https://obamawhitehouse.archives.gov/the-press-office/2015/07/29/

executive-order-creating-national-strategic-computing-initiative.

[3] 肖飞，张为华，王东辉. 面向科学过程的工作流技术研究现状与趋势 [J]. 计算机应用研究, 2011, 28 (11): 4013-4019. (Xiao Fei, Zhang Weihua, Wang Donghui. Overview of workflow technology in scientific process [J]. Application Research of Computers, 2011, 28 (11): 4013-4019.)

[4] Bertram L, Berkley A C, *et al.* Scientific workflow management and the Kepler system [J]. Concurrency and Computation: practice & Experience. 2006, 18: 1039-1065.

[5] Deelman E, Singh G, Su M H, *et al.* Pegasus: a framework for mapping complex scientific workflows onto distributed systems [J]. Scientific Programming, 2005, 13 (3): 219-237.

[6] Wolstencroft K, Haines R, Fellows D, *et al.* The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud [J]. Nucleic Acids Research, 2013, 1 (1): 1–5.

[7] 王红霞. 网格工作流引擎的设计与实现 [J]. 计算机工程与设计，2011, 32 (2): 430-433. (Wang hongxia. Design and realization of grid workflow engine [J]. Computer Engineering and Design, 2011, 32 (2): 430-433.)

[8] 沈瑜，李娟，常飚，等. 高性能计算机统一资源管理系统的设计与实现 [J]. 计算技术与自动化, 2014, 33 (1): 83-90. (Shen Yu, Li Juan, Chang Biao, *et al.* Design and Implementation of the Uniform Resource Management System of HPC [J]. Computing Technology and Automation, 2014, 33 (1): 83-90.)

[9] Topcuoglu, Hariri, Y Wu. Performance effective and low complexity task scheduling for heterogeneous computing [J] , IEEE Trans on Parallel and Distributed System, 2002, 13 (3): 260-274.

[10] Shi Zhiao, Dongarra J J. Scheduling workflow applications on processors with different capabilities [J]. Future Generation Computer Systems 2006, 22: 665–675.

[11] Kwok Y K, Ahmad. Dynamic critical-path scheduling an effective technique for allocating task graphs to multiprocessors [J]. IEEE Trans on Parallel and Distributed System, 1996, 7 (5): 506-521.

[12] Venugopal R, Buyya R. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids [C]// Proc of IEEE International Conference on e-Science and Grid Computing. 2008: 35-42.

[13] Rafiul R, Hassan M, Ranjan R, *et al.* Adaptive workflow scheduling for dynamic grid and cloud computing environment [J]. Concurrency and Computation: practice & Experience, 2013, 25: 1816–1842.

[14] Chan W Y, Li C K. Heterogeneous dominant sequence cluster (HDSC): a low complexity heterogeneous scheduling algorithm [C]// Proc of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. 1997.

[15] Amalarethinam G, Selvi F K M. A minimum makespan grid workflow scheduling algorithms [C]// Proc of Conference on Computer Communication and Informatics. 2012.

[16] Patil V A, Chaudhary V. Rack aware scheduling in HPC data centers: an energy conservation strategy [J]. Cluster Computing, 2013, 16 (3): 559-573.

[17] Chen Wei, Lee Y C, Fekete A, *et al.* Adaptive multiple-workflow scheduling with task rearrangement [J]. The Journal of Supercomputing, 2015, 71 (4): 1297-1317.

[18] Wu Fuhui, Wu Qingbo, Tan Yusong. Workflow scheduling in cloud: a survey [J]. Journal of Supercomputing, 2015, 71 (9): 1-46.

[19] Frederic Suter. DAGGEN [EB/OL]. (2017-07-26) [2018-05-08]. https://github. com/frs69wq/daggen.

[20] Casanova H, Giersch A, Legrand A, *et al.* Versatile, scalable, and accurate simulation of distributed applications and platforms [J]. Journal of Parallel and Distributed Computing, 2014, 74 (10): 2899-2917.